EMBEDDED
LINUX
CONFERENCE

Engineering Secure SSH Access for Engineers

Colin McAllister, Garmin Ltd.

THE LINUX FOUNDATION

#EmbeddedOSSummit @colin-pm

Colin McAllister

- Software Engineer at Garmin
    - Assists in developing Garmin Marine's Linux distributions
- Worked with Embedded Linux for about 6 years
- Open-source contributor
    - OpenEmbedded, Yocto meta-layers, swupdate, and HVAC
- When I'm not working, you can find me on a bicycle

Disclaimer

- I am not a dedicated cybersecurity professional
- I am an Embedded Linux developer who spent a lot of time to researching different SSH authentication methods
    - Want to share what I learned with the community
- This is not a detailed discussion of how SSH authentication methods work
    - Want to focus discussing methods and how they comparatively meet the challenges of Embedded Linux devices
- The views expressed within this presentation are those of the presenter; they do not necessarily reflect the views of Garmin

The Secure Shell

The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network

This document describes the SSH transport layer protocol, which typically runs on top of TCP/IP. The protocol can be used as a basis for a number of secure network services. It provides strong encryption, server authentication and integrity protection. It may also provide compression.

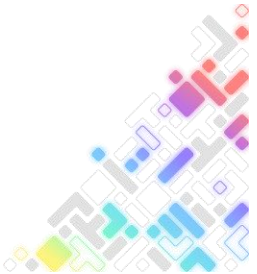
RFC 4253

SSH is a Swiss Army Knife for developers

- Remote shell access
- Tunneling and port forwarding
- File transfer
- X11 forwarding
- All over an encrypted and authenticated connection

SSH Authentication

- The SSH daemon (sshd) supports multiple ways for clients to authenticate
- Authentication methods can be enabled, disabled, and configured within the sshd configuration file
    - /etc/ssh/sshd
    - Requires root-level permissions to edit
    - Includes default authentication configurations



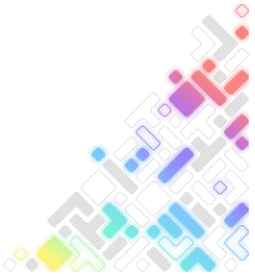
SSH & Embedded Linux Devices

- Typically provides root access for developers
- Embedded devices are the SSH server
- Embedded Linux products may not have…
  - Internet connectivity
  - Correct system time
- Software updates may be difficult or not guaranteed to be timely

Rackmount-guy at English Wikipedia

Embedded Linux Security

- Security is critical for Embedded Linux products
- Embedded devices often have a physical presence
  - Industrial, automotive, marine, medical, etc.
  - Possible direct impact on user safety
- Even non-safety critical products have security considerations
  - Conference Room Controllers
  - Baby monitors
- Governments are starting to make security a requirement
  - EU Radio Equipment Directive (RED)
  - US Cyber Trust Mark



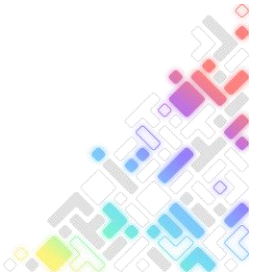
Most Secure Solution

- Turn off SSH on release builds
- Eliminates any risk of third parties gaining root access over SSH
- However…
  - Prevents developers from interacting with builds released to customers
  - Prevents any sort of field diagnostics and maintenance

Need a secure authentication method that works with deployed devices

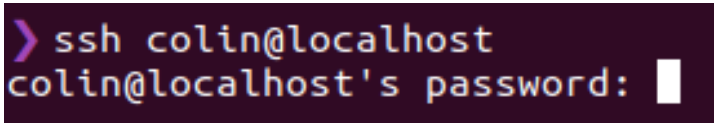
Authentication Requirements for Embedded Linux

- Solution must be secure and limit access to only approved individuals
- Solution must easily scale to many devices
- Authentication may be required without internet connectivity
- Authentication may be required without correct system time



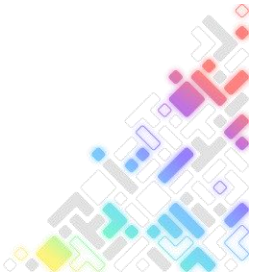
Password Authentication

- The default SSH authentication mechanism
- Client prompted to enter the remote user's password
- Two options exist when using password authentication with multiple hosts
  - Shared password used across all hosts
  - Each host configured with a unique password



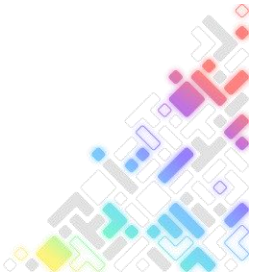
Shared & Unique Passwords

- Shared passwords
  - Baked into firmware and updatable within firmware updates
  - Compromised password affects all devices
- Unique passwords
  - Requires database or password derivation function
  - Compromised password impacts only a single device
  - Requires configuration at runtime
  - Hard to update passwords
  - Tricky for developers to obtain passwords



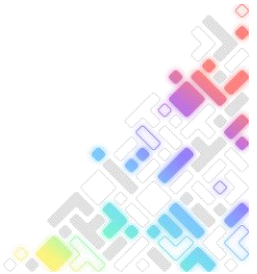
Don’t use password authentication

- Passwords can be cracked
- A compromised password leaves one or more devices compromised
- Shared passwords prohibited by regulators
- Unique passwords add more complexity than security
  - Database or password derivation function could become compromised
  - Clients require internet to access password database
  - Function also subject to exfiltration or reverse engineering
  - Hard to update passwords
- Hard to keep passwords secret
  - No built-in method to automate password entry



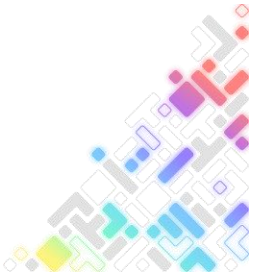
Public Key Authentication

- Automated challenge response mechanism
  - Server sends client a challenge
  - Client generates response with their private key
  - Server verifies response with their copy of client's public key
- Trusted public keys are stored on host in text database file
  - $HOME/.ssh/authorized_keys



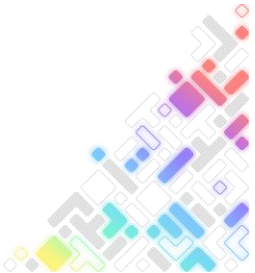
Public Key Authentication

- Private keys are harder to crack versus passwords
- SSH users will need filesystem access to private key
  - Subject to exfiltration
- Suffers same shared vs. unique problem as password authentication; either:
  - All devices trust same key pair
  - Must create and manage unique keys for each device



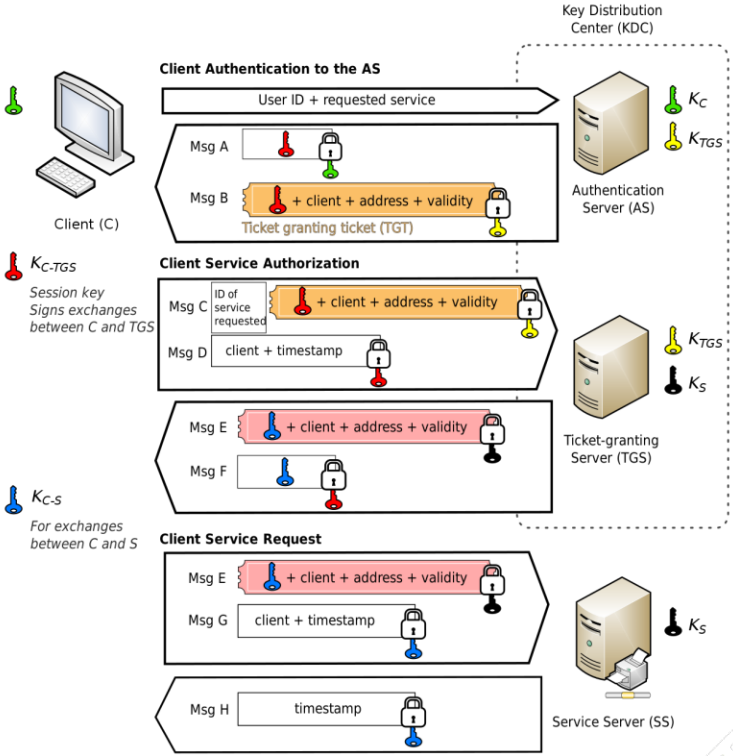
Avoid using public key authentication

- Although keys can be more secure than passwords
- Suffers same issues as passwords
  - Risk using a single key
  - Manage complexity of unique keys for each device



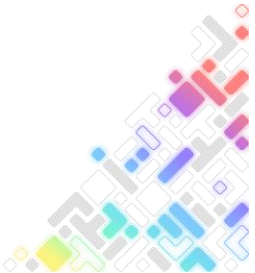
GSSAPI / Kerberos Authentication

- More complex authentication mechanism
  - Authentication and ticket granting services
  - All clients and devices share symmetric keys with services
  - Client must authenticate and gain ticket to establish connection with device



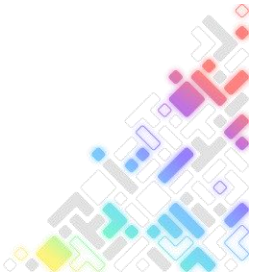
GSSAPI / Kerberos Authentication

- Infrastructure supports scaling
- Use of symmetric keys provides strong security
- Tricky to use in an embedded context
  - All devices and infrastructure require accurate time
  - Need to manage unique keys for each client & device
    - Difficult to initialize and update on edge devices
- Strong authentication method, but likely difficult to implement for embedded uses



Certificate-based Authentication

- Devices all trust public key of certificate authority (CA)
  - CA private keys protected within signing server
- All clients can use their own key pairs
- Client's public key must be signed by CA to allow client key pair to be accepted by host
  - Authentication can be used with the signing server
- Host verifies client's signed public key with the CA's public key



SSH Certificates

- Certificate format unique to SSH
- Signed public key with…
  - Validity period
  - Critical Options
    - Control features that restrict access
    - force-command, source-address, verify-required
  - Extensions
    - Enables features that grant access
    - permit-x11-forwarding, permit-port-forwarding, etc.

```
❯ ssh-keygen -L -f client_id_ecdsa-cert.pub
client_id_ecdsa-cert.pub:
        Type: ecdsa-sha2-nistp256-cert-v01@openssh.com user certificate
        Public key: ECDSA-CERT SHA256:FRigp+3O4lbV0Ba+VdoAoAH519zlgUK9tz5aqek+PDQ
        Signing CA: ECDSA SHA256:hpU8/r7ed5C5QqDBeQU0JXcD32LI1psF7WW7JP/AaCs (using ecdsa-sha2-nistp256)
        Key ID: "dev@company.com"
        Serial: 0
        Valid: from 2024-04-14T17:42:00 to 2024-04-15T17:43:31
        Principals:
                colin
        Critical Options: (none)
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

Certificate Authentication Demo

- Create CA key pair

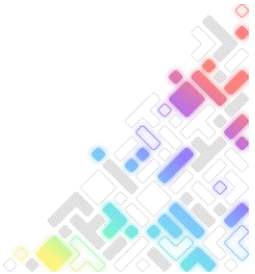
  ssh-keygen –f ca_id_ecdsa –t ecdsa -q –N “”
- Create client’s key pair

  ssh-keygen –f client_id_ecdsa –t ecdsa -q –N “”
- Add CA's public key to host's authorized keys file

  echo "TrustedUserCAKeys /etc/ssh/ca_id_ecdsa.pub" > /etc/ssh/sshd_config
- Get client certificate

  ssh-keygen -s ca_id_ecdsa -I dev@company.com -n root -V +1d client_id_ecdsa.pub
- Use SSH with certificate

  ssh -i client_id_ecdsa root@embedded_device.local.



Dealing with unreliable device time

- Checking the certificate validity period may fail if host clock is off
- OpenSSH can be easily patched to disable where the validity period is checked in sshkey.c
- However, a certificate will now always be valid
  - Leaves solution no better than public key authentication

```
2327    if (verify_time < k->cert->valid_after) {
2328        *reason = "Certificate invalid: not yet valid";
2329        return SSH_ERR_KEY_CERT_INVALID;
2330    }
2331    if (verify_time >= k->cert->valid_before) {
2332        *reason = "Certificate invalid: expired";
2333        return SSH_ERR_KEY_CERT_INVALID;
2334    }
```

Adding a new critical option

- Assume we have devices where each hostname includes UUID
- Let’s create a new critical option that contains a device’s identity
  - “hostname@company.com”
- sshd can be patched to only accept certificates when the device identity matches the critical option
- Binds certificates over the device identity domain
- When used in addition to disabling validity period checking, a certificate now provides indefinite access to only a single device

Demo: Adding a new critical option

- Patch OpenSSH to verify "hostname@company.com"
- Changes made in auth-options.c
- sshd will now check for new critical option

```
--- a/auth-options.c
+++ b/auth-options.c
@@ -30,6 +30,7 @@
 #include <stdarg.h>
 #include <ctype.h>
 #include <limits.h>
+#include <unistd.h>

 #include "openbsd-compat/sys-queue.h"

@@ -79,6 +80,8 @@ cert_option_list(struct sshauthopt *opts, struct sshbuf *oblob,
 {
    char *command, *allowed;
    char *name = NULL;
+   char *hostname = NULL;
+   char expected_hostname[HOST_NAME_MAX];
    struct sshbuf *c = NULL, *data = NULL;
    int r, ret = -1, found;
```

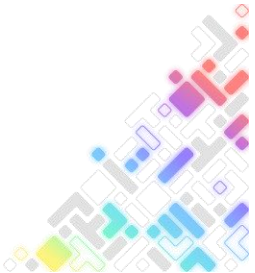
```
@@ -161,6 +164,30 @@ cert_option_list(struct sshauthopt *opts, struct sshbuf *oblob,
                }
                opts->required_from_host_cert = allowed;
                found = 1;
+           } else if (strcmp(name, "hostname@company.com") == 0) {
+               if (hostname != NULL) {
+                   error("Certificate has multiple "
+                       "hostname@company.com critical options");
+                   goto out;
+               }
+               if ((r = sshbuf_get_cstring(data, &hostname,
+                   NULL)) != 0) {
+                   error_r(r, "Unable to parse \"%s\" "
+                       "section", name);
+                   goto out;
+               }
+               if (gethostname(expected_hostname, HOST_NAME_MAX)) {
+                   error("Unable to get hostname");
+                   free(hostname);
+                   goto out;
+               }
+               if(strncmp(hostname, expected_hostname, HOST_NAME_MAX) != 0) {
+                   logit("Authentication tried on device with hostname \"%s\""
+                   " but certificate limits use to \"%s\"", expected_hostname, hostname);
+                   free(hostname);
+                   goto out;
+               }
+               found = 1;
```

Demo: Adding a new critical option

- Sign public key with extra critical option

  ssh-keygen -s user_ca -I dev@company.com -n root -V +1d \

  -O critical:hostname@company.com=mydevice-123 client_id_ecdsa.pub
- Certificate will now only authenticate with mydevice-123.local.
- Signing server can force the use of this critical option



Certificate Authentication

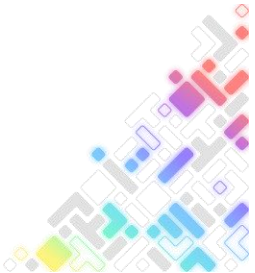
- Easy to configure hosts to trust single CA public key
  - Firmware update can roll CA public key on devices
- CA key can be stored securely
  - Authentication required to sign public keys
  - Can audit each signing operation
  - Signing server implementations already exist
    e.g., HashiCorp Vault and Smallstep SSH
- Certificates can be short lived if time is guaranteed
- Critical options and extensions add extra flexibility and features
  - Certificates can be limited to be used with a single device
- Easy to alter certificate authentication in OpenSSH to support embedded requirements
- Can use host certificates in addition to client certificates!

PAM Authentication

- PAM = Pluggable Authentication Module
- Allows sshd to be configured to support more authentication modules than what OpenSSH supports out of the box
- Can use authentication modules specified in the host's PAM configuration files
  - OAuth 2.0, Google Authenticator, Yubikey, etc.
- Lots of options to explore if none of the solutions discussed today seem satisfactory

Thank you for your time

- Questions?



References

- OpenSSH Certificate Standard
- OpenSSH Patches
  - Disabling time
  - Adding hostname critical option
- SSH CA Implementations
  - HashiCorp Vault
  - Smallstep SSH
- PAM Modules
  - Oauth 2.0
  - Google Authenticator
  - Yubikey
- Regulations
  - EU RED
  - US Cyber Trust Mark
  - No shared passwords part of ETSI EN 303 645 (page 13 - Clause 5.1)

EMBEDDED LINUX CONFERENCE